



```
pip install dgl -f https://data.dgl.ai/wheels/torch-2.3/cu121/repo.html
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import torch
import torch.nn as nn
import torch.nn.functional as F
import dgl
import os
from dgl.nn import GraphConv, HeteroGraphConv
import numpy as np
import torch.optim as optim
from torch.optim import Adam
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset
from dgl.data.loading import GraphDataLoader
import dgl.nn as dglnn
from dgl.nn import GATConv
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
```

 Using device: cuda

```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

```
dirve_path = '/content/drive/MyDrive/Project/'
train = pd.read_csv(dirve_path + 'security_train.csv')
```

Diagram

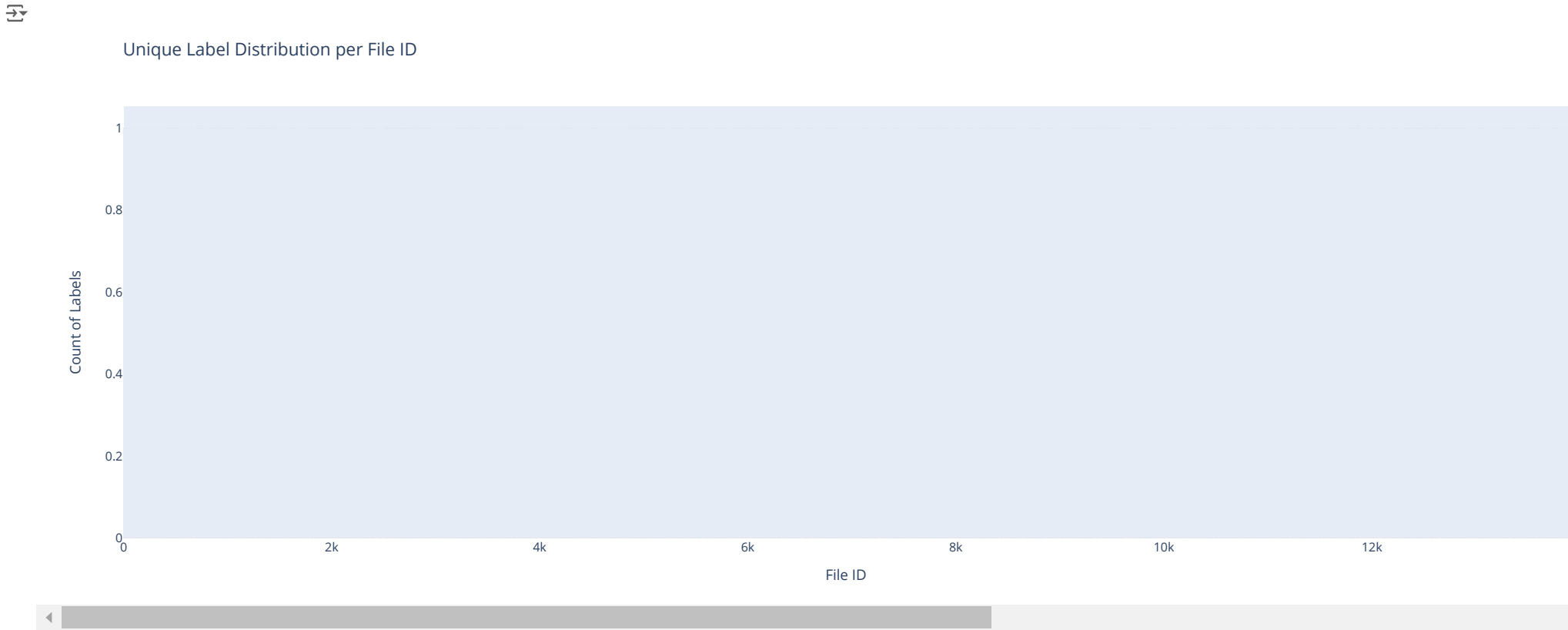
```
df = train
```

```
# Sort by file_id and label
df = df.sort_values(by=['file_id', 'label'])

# Remove duplicate file_id and label pairs, keeping only the first occurrence of the record
df_unique = df.drop_duplicates(subset=['file_id', 'label'], keep='first')
```

```
df_count = df_unique.groupby(['file_id', 'label']).size().reset_index(name='count')
```

```
# Use Plotly to draw interactive charts
fig = px.bar(df_count, x='file_id', y='count', color='label',
             title='Unique Label Distribution per File ID',
             labels={'count': 'Count of Labels', 'file_id': 'File ID', 'label': 'Label'},
             height=600)
fig.show()
```



HIN

```
# Optimise data types to reduce memory usage
train['tid'] = train['tid'].astype('int32')
train['index'] = train['index'].astype('int32')
train['api_idx'] = train['api'].astype('category').cat.codes
train.sort_values(by=['tid', 'index'], inplace=True)
train['next_api_idx'] = train.groupby('tid')['api_idx'].shift(-1)
```

```
file_ids = train['file_id'].unique()
train_file_ids, val_file_ids = train_test_split(file_ids, test_size=0.2, random_state=42)
train_data = train[train['file_id'].isin(train_file_ids)]
val_data = train[train['file_id'].isin(val_file_ids)]
```

```
global_api_idx_range = np.arange(train['api_idx'].min(), train['api_idx'].max() + 1)
global_tid_range = np.arange(train['tid'].min(), train['tid'].max() + 1)
```

```
def adjust_features_to_nodes(features, num_nodes):
    num_features = features.shape[0]
    if num_features < num_nodes:
        # If there are not enough features, repeat the last feature until the number of matching nodes is reached
        if num_features > 0: # Ensure that at least one feature is available for repetition
            last_feature = features[-1, :].unsqueeze(0)
            additional_features = last_feature.repeat(num_nodes - num_features, 1)
            adjusted_features = torch.cat([features, additional_features], dim=0)
        else: # If there are no features, create an all-zero feature matrix
            adjusted_features = torch.zeros((num_nodes, 1)) # Assuming a feature dimension of 1
    elif num_features > num_nodes:
        # If there are too many features, take the first num_nodes of features
        adjusted_features = features[:num_nodes]
    else:
        # The number of features matches exactly
        adjusted_features = features
    return adjusted_features
```

```
def create_hetero_graphs(data, global_api_idx_range, global_tid_range):
    # Construct a dictionary of graphs, grouped by file_id
    list_of_graphs = []
    list_of_labels = []

    for file_id, file_data in data.groupby('file_id'):
        # Calculate the edge
        file_thread_edges = file_data[['file_id', 'tid']].drop_duplicates()
        thread_api_edges = file_data[['tid', 'api_idx']].drop_duplicates()
        #num_apis = thread_api_edges['api_idx'].nunique()

        # Calculate API call order edges
        valid_next_edges = file_data.dropna(subset=['next_api_idx'])
        next_edges = pd.DataFrame({
            'src': valid_next_edges['api_idx'].values,
            'dst': valid_next_edges['next_api_idx'].values.astype('int')
        })

        # Construct heterogeneous maps
        graph = dgl.heterograph({
            ('file', 'contains', 'thread'): (torch.tensor(file_thread_edges['file_id'].values, dtype=torch.long),
                                              torch.tensor(file_thread_edges['tid'].values, dtype=torch.long)),
            ('thread', 'calls', 'api'): (torch.tensor(thread_api_edges['tid'].values, dtype=torch.long),
                                         torch.tensor(thread_api_edges['api_idx'].values, dtype=torch.long)),
            ('api', 'next', 'api'): (torch.tensor(next_edges['src'].values, dtype=torch.long),
                                    torch.tensor(next_edges['dst'].values, dtype=torch.long))
        })

        # Add file node features
        num_files = len(file_thread_edges['file_id'].unique()) # of file nodes
        file_features = torch.tensor(file_data[['tid', 'api_idx']].nunique().values).float().reshape(-1, 1)
        #print("file_features",file_features)
        num_nodes = graph.number_of_nodes('file')
        #print("Number of 'file' nodes in the graph:", num_nodes)
        file_features_adjusted = adjust_features_to_nodes(file_features, num_nodes)
        graph.nodes['file'].data['feat'] = file_features_adjusted

        # Calculate API node characteristics
        #api_counts = file_data['api_idx'].value_counts().reindex(global_api_idx_range, fill_value=0).sort_index().values
        api_counts = file_data['api_idx'].value_counts().sort_index().values
        api_features = torch.tensor(api_counts, dtype=torch.float).unsqueeze(1)
        #print(api_counts, len(api_counts))
        num_api_nodes = graph.number_of_nodes('api')
        #print("Number of 'api' nodes in the graph:", num_api_nodes)
        api_features_adjusted = adjust_features_to_nodes(api_features, num_api_nodes)
        graph.nodes['api'].data['feat'] = api_features_adjusted

        # Add diagram tags
        label = file_data['label'].iloc[0] # Assume all rows with the same file_id have the same label
        list_of_labels.append(label)
        # Add the diagram to the dictionary
        list_of_graphs.append(graph)

    batched_graph = dgl.batch(list_of_graphs)
    batched_labels = torch.tensor(list_of_labels)
    return batched_graph, batched_labels
```

```
train_graph, train_labels = create_hetero_graphs(train_data, global_api_idx_range, global_tid_range)
```

```
val_graph, val_labels = create_hetero_graphs(val_data, global_api_idx_range, global_tid_range)
```

```
train_graph
```

```
🔗 Graph(num_nodes={'api': 3139502, 'file': 77279247, 'thread': 30928397},
        num_edges=({'api', 'next', 'api'): 71801067, ('file', 'contains', 'thread'): 207040, ('thread', 'calls', 'api'): 3196983},
        metagraph=[('api', 'api', 'next'), ('file', 'thread', 'contains'), ('thread', 'api', 'calls')])
```

```
train_labels
```

```
🔗 tensor([5, 2, 0, ..., 0, 1, 2])
```

```
val_graph,
```

```
🔗 (Graph(num_nodes={'api': 783684, 'file': 19165968, 'thread': 7726490},
        num_edges=({'api', 'next', 'api'): 18002844, ('file', 'contains', 'thread'): 54004, ('thread', 'calls', 'api'): 803216},
        metagraph=[('api', 'api', 'next'), ('file', 'thread', 'contains'), ('thread', 'api', 'calls')]),)
```

```
val_labels
```

```
🔗 tensor([0, 0, 5, ..., 0, 0, 2])
```

```
def save_graphs(graph, filename):
    dgl.save_graphs(filename, [graph])
```

```
def load_graphs_from_file(filename):
    graphs, _ = dgl.load_graphs(filename)
    return graphs
```

```
def save_labels(labels, filename):
    torch.save(labels, filename)
```

```
def load_labels_from_file(filename):
    loaded_labels = torch.load(filename)
    return loaded_labels
```


```
train_graphs_filename = '/content/drive/My Drive/Graphs/results/train_hetero_graphs_3_edges.bin'
val_graphs_filename = '/content/drive/My Drive/Graphs/results/val_hetero_graphs_3_edges.bin'
train_labels_filename = '/content/drive/My Drive/Graphs/results/train_hetero_graphs_3_edges.pth'
val_labels_filename = '/content/drive/My Drive/Graphs/results/val_hetero_graphs_3_edges.pth'
```

```
save_graphs(train_graph, train_graphs_filename)
save_graphs(val_graph, val_graphs_filename)
```

```
save_labels(train_labels, train_labels_filename)
save_labels(val_labels, val_labels_filename)
```

```
train_graph = load_graphs_from_file(train_graphs_filename)
val_graph = load_graphs_from_file(val_graphs_filename)
```


train_graph, val_graph

 (Graph(num_nodes={'api': 3139502, 'file': 77279247, 'thread': 30928397}, num_edges=({'api', 'next', 'api'): 71801067, ('file', 'contains', 'thread'): 207040, ('thread', 'calls', 'api'): 3196983}, metagraph=[('api', 'api', 'next'), ('file', 'thread', 'contains'), ('thread', 'api', 'calls')]), Graph(num_nodes={'api': 783684, 'file': 19165968, 'thread': 7726490}, num_edges=({'api', 'next', 'api'): 18002844, ('file', 'contains', 'thread'): 54004, ('thread', 'calls', 'api'): 803216}, metagraph=[('api', 'api', 'next'), ('file', 'thread', 'contains'), ('thread', 'api', 'calls')]))

```
train_graph = train_graph[0]
val_graph = val_graph[0]
```

```
train_labels = load_labels_from_file(train_labels_filename)
val_labels = load_labels_from_file(val_labels_filename)
```

train_labels, val_labels

 (tensor([5, 2, 0, ..., 0, 1, 2]), tensor([0, 0, 5, ..., 0, 0, 2]))

```
class GraphDataset(Dataset):
    def __init__(self, graphs, labels):
        self.graphs = graphs # It's a list of diagrams
        self.labels = labels # It's a list of tags that correspond to the diagram

    def __len__(self):
        return len(self.graphs)

    def __getitem__(self, idx):
        return self.graphs[idx], self.labels[idx]
```

```
individual_train_graphs = dgl.unbatch(train_graph)
individual_val_graphs = dgl.unbatch(val_graph)
```

```
train_dataset = GraphDataset(individual_train_graphs, train_labels)
train_dataloader = GraphDataLoader(train_dataset, batch_size=4, shuffle=True)
```

```
val_dataset = GraphDataset(individual_val_graphs, val_labels)
val_dataloader = GraphDataLoader(val_dataset, batch_size=4, shuffle=True)
```